Group Members:

Tanveer Ahmed (FA20-BCS-063)

Usama Pervez(SP20-BCS-013)

Q2: Functionalities along with screenshots?

1. Tokenization:

The primary functionality of the code is to tokenize input source code. Tokenization is the process of breaking down the source code into meaningful units called tokens. Tokens represent the basic building blocks of a programming language, including identifiers, literals, operators, and other language elements. The `Lexer` class in the code is responsible for reading the input code character by character, identifying and categorizing each token, and providing information about its type and value. This functionality is crucial in the early stages of compiler construction for further syntactic and semantic analysis.

2. User Interaction and Output Display:

The code includes a simple user interface for interaction. The `Main` method in the `Program` class prompts the user to input code through the console. The lexer (`Lexer` class) then processes the input code, tokenizes it, and displays the results in the console. This user interaction facilitates the testing and demonstration of the lexer's functionality. The program continues tokenizing the input until the end-of-file token is encountered, and the results are displayed, providing insights into how the lexer interprets the given code. The user interface functionality enhances the usability of the lexical analyzer, making it accessible for testing and educational purposes.

Screenshots:

```
ar.IsDigit(currentChar))

ReadNumber();

ew Except
```

```
C:\Users\student\source\repos\hiovfshgoih\hiovfshgoih\bin\Debug\hiovfs
Enter your code:
let sum = 0;
Type: Assignment, Value: =
Type: Identifier, Value: sum
Type: Assignment, Value: =
Type: IntegerLiteral, Value: 0
```

```
C:\Users\student\source\repos\hiovfshgoih\hiovfshgoih\bin\Debug
Enter your code:
let result = a * b + 7;
Type: Assignment, Value: =
Type: Identifier, Value: result
Type: Assignment, Value: =
Type: Identifier, Value: a
Type: Multiply, Value: *
Type: Identifier, Value: b
Type: Plus, Value: +
Type: IntegerLiteral, Value: 7
Type: Semicolon, Value: ;
Type: EndOfFile, Value:
Enter your code:
let result = a * b + 7;
Type: Assignment, Value: =
Type: Identifier, Value: result
Type: Assignment, Value: =
Type: Identifier, Value: a
Type: Multiply, Value: *
Type: Identifier, Value: b
Type: Plus, Value: +
Type: IntegerLiteral, Value: 7
Type: Semicolon, Value: ;
Type: EndOfFile, Value:
Enter your code:
```

Tokenization:

```
1 reference
public Token GetNextToken()
{
    SkipWhitespace();

    if (currentPosition >= input.Length)
    {
        return new Token(TokenType.EndOfFile, "");
    }

    char currentChar = CurrentChar;

    switch (currentChar)
    {
        case '+':
            MoveNext();
            return new Token(TokenType.Plus, "+");

        case '-':
            MoveNext();
            return new Token(TokenType.Minus, "-");

        case '*':
            MoveNext();
            return new Token(TokenType.Multiply, "*");

        case '/':
            MoveNext();
            return new Token(TokenType.Divide, "/");
```